Indexy

Indexy

- Pokud bychom vytvořili tabulku, naplnili ji daty a potřebovali bychom vyhledat určitý záznam (např. podle id zaměstnance):
  - Záznamy v tabulce jsou ukládány neuspořádaně, nejčastěji v pořadí, v jakém byly do tabulky vkládány,
  - záznamy se vyhledávájí od prvního záznamu postupně, dokud se hledaný nenajde,
  - v průměru dojde k prohledání poloviny záznamů – časově nákladná operace.
- Tato situace se vyřeší přidáním dalšího databázového objektu – indexu.
- Index obsahuje informaci, kde je jaký záznam uložen (pro hodnoty prohledávaných sloupců eviduje rowID – unikátní číslo záznamu v databázové tabulce, interně vytvářené a spravované databází).

Indexy

- Indexy jsou volitelnou strukturou asociovanou s tabulkou.
- Index se tvoří nad jedním či více sloupci tabulky.
- Složený index (composite index, případně také concatenated index) je index složený z více sloupců v tabulce. Na pořadí sloupců v indexu záleží.
- Index může být vytvořen i nad stejnými sloupci jako jiný existující index, pokud jsou sloupce v indexech uvedeny v rozdílném pořadí.
- Oracle automaticky vytváří indexy na sloupcích, které jsou primárním klíčem nebo jsou unique.

Kompozitní indexy

- Kompozitní indexy obsahují více než jeden sloupec a mohou být výkonnější než indexy, obsahující pouze jeden sloupec.
- Kde použít kompozitní indexy? Nad klíči, které jsou často použity společně ve WHERE klauzuli a jsou spojené pomocí AND operátoru.
- Záleží na pořadí, v jakém jsou sloupce v kompozitním indexu uvedeny? Pokud má jeden z indexů výrazně více rozdílných hodnot (nejvíc unikátních hodnot), pak ano … a je vhodné ho uvést jako první.

Indexy

- Při hledání podle indexového sloupce se v indexu najde uzel, ve kterém se přečte rowID (jedinečný identifikátor záznamu v tabulce – odkaz do tabulky) a podle něj se již velmi rychle (s konstantní složitostí) vrátí požadovaný záznam v tabulce.
- Databázové servery obvykle používájí stromovou strukturu indexů (například B-stromy).
- Vyhledávání v uspořádané stromové struktuře je v průměru výrazně rychlejší než sekvenční vyhledávání v neuspořádaném poli.

Základní typy indexů

- **Normální index** – výchozí, nejčastěji používaná varianta (interně např. B-strom).
- **Bitmapový index** – interně reprezentovaný jako bitmapa. Pro každou možnou hodnotu indexového sloupce se eviduje pole bitů. Každý bit v poli bitů odpovídá možnému rowID. Pokud je bit nastaven, pak odpovídající řádek obsahuje klíčovou hodnotu. Používá se pro sloupce s nízkou variabilitou (malé množství možných hodnot).
  Př. Pro sloupec „pohlaví“ jsou nutná dvě pole bitů (muž, žena).
  Bitmap index je ale pouze v Oracle Enterprise Edition!
- **Funkční index** – založený na funkcích a výrazech, které mohou zahrnovat jeden či více sloupců v tabulce. Funkční index vypočítá hodnotu funkce či výrazu a uloží ji do indexu. K této hodnotě je přiřazováno odpovídající rowID. Funkční index lze vytvořit jako B-strom i jako bitmapu.

Vlastnosti indexu

- Hledání pomocí indexu je výhodné v případě, že je potřeba nalézt pouze několik záznamů. Pokud je třeba vyhledat více jak 5% záznamů z tabulky, je obvykle rychlejší sekvenční prohledávání bez použití indexu.
- Není doporučené používat indexy nad tabulkami s malým obsahem dat.
- Přidání/odstranění indexu neovlivňuje data uložená v tabulce.
- Při ORDER BY se index nepoužije, když jsou v něm všechny sloupce NULLable!

Režie indexu

- Při použití indexu vzroste rychlost vyhledávání podle indexovaných sloupců, ale vzroste také režie při vkládání a změně dat v tabulce. Při vložení nebo editaci dat je potřeba zapsat hodnoty nejen do tabulky, ale také do indexu!
- Každý index zvyšuje režii DML operací (INSERT, UPDATE, DELETE) cca. trojnásobně!
- Pokud zavoláte INSERT na tabulku se třemi indexy, pak bude tato operace cca. 10x pomalejší než kdyby indexy neměla!
- U tabulek, do kterých se hodně ukládají data je nutné volit kompromis mezi výkonem SELECTů a INSERT operací!

Unikátní a NULLové hodnoty

- Index může být jednoznačný (unique) či víceznačný (nonunique). Jednoznačný index můžete použít, pokud v tabulce nemůže existovat více řádků, které by pro indexové sloupce měly stejné hodnoty.
  - UNIQUE index je rychlejší než NONUNIQUE.
  - POZOR! Pokud vytvoříte UNIQUE index například nad kombinací sloupců (např. JMENO, PRIJMENI), pak tato kombinace skutečně musí být vždy unikátní! Jinak Oracle nepovolí změny (INSERT / UPDATE operace), které by tuto podmínku porušily!!!
- Hodnoty NULL v indexech jsou považovány za různé vyjma případu, kdy všechny neNULLové hodnoty ve dvou či více sloupcích indexu jsou shodné. V tom případě jsou řádky považovány za identické.

Index & IS NULL

- K tomu, aby se použil index u IS NULL podmínky, nestačí jenom že ve sloupci nejsou NULL hodnoty, sloupec musí vyloženě být NOT NULL.
- Další problém je, když chceme zjistit, jestli je výsledek skalární funkce NULL. Oracle nemůže vědět jestli funkce vrátí NULL a proto nemůže použít index.
  - Kromě pár výjimek jako UPPER(), u kterých Oracle ví že nemohou vracet NULL když jsou aplikovány na NOT NULL sloupec.
- Toto jsou všechno důvody, proč bychom v databázi neměli mít NULL hodnoty vůbec (pokud pak chceme v SELECTu zjišťovat, jestli tam NULL je nebo není).

Vytvoření indexu

```
CREATE [UNIQUE | BITMAP] INDEX <název indexu>
ON <název tabulky> (<seznam sloupců>);
```

- Normální index:
```
CREATE INDEX ix_zamestnanec
ON zamestnanec (prijmeni, jmeno);
```
- Bitmapový index:
```
CREATE BITMAP INDEX ix_rodinny_stav
ON zamestnanec (rodinny_stav);
```
- Funkční index:
```
CREATE INDEX ix_prijmeni
ON zamestnanec (UPPER (prijmeni));
```
- Smazání indexu: `DROP INDEX <název indexu>;`

Funkční index

- Budeme optimalizovat tento dotaz:

```
select * from cities where lower(full_name_nd)
like 'new york%';
```

- Vyzkoušejte délku trvání dotazu bez a s tímto indexem:

```
create index i_cities_lower
on cities (lower(full_name_nd));
```

- U tohoto dotazu není možné se vyhnout full table / index scan:

```
select * from cities where lower(full_name_nd)
like '%new york%';
```

Pro fulltextové prohledávání je mnohem lepší využít Elastic Search / Solr / Lucene nebo nativní funkcionalitu databází.

- Funkční index použijete pro:
  - case-insensitive vyhledávání (viz. předchozí příklad).
  - dotazy nad záznamy na základě vypočítané hodnoty například v databázi eshopu můžete chtít najít objednávky, které firmě přinesly nejvíc peněz: (cena – sleva) * množství

Indexy – shrnutí

- Indexy je vhodné vytvořit:
  - V tabulkách obsahujících velké množství dat pro urychlení vyhledávání, pokud je potřeba vyhledávat malé množství záznamů v tabulce (méně jak 5%),
  - pro urychlení spojování tabulek.
- Vhodné atributy pro indexování:
  - Atributy, podle kterých se vyhledává (jsou ve WHERE, spojují se podle nich tabulky, provádí se podle nich řazení),
  - vysoká variabilita hodnot – normální index,
  - nízká variabilita hodnot (jen několik možných hodnot) – bitmapový index.
- Existující index je vhodné smazat pokud:
  - Neurychluje výsledky dotazů,
  - dotazy používané v aplikaci index nepoužívají,
  - je-li nutné index znovu postavit.

ORDER

- Při vytváření indexu je možné vybrat, v jakém pořadí v něm budou uloženy záznamy (ASC, DESC). Toto nastavení se hodí při řazení, protože Oracle nebude muset provést řazení záznamů získaných s pomocí indexu. V posledních verzích ale Oracle preferuje INDEX FULL SCAN a toto pořadí ignoruje.

ORDER BY

- Indexy je také vhodné využívat při použití ORDER BY.

Indexy & datumy

- Pokud chcete například získat všechny záznamy z roku 2000, pak můžete použít select jako:

```
select * from tabulka
   where extract (year from datum) = 2000
```

- ALE: Nesmíte zapomenout na nastavení funkčního indexu
- Bez funkčního indexu to samé:

```
select * from zamestnanec
   where datum_nastupu >= '1.1.2000'
   and datum_nastupu < '1.1.2001';
```

- Obdobně je problematické použití funkce TRUNC(), která se často používá k získání samotného datumu (protože v Oracle DATE obsahuje datum i čas).

Indexy & …

- Obdobným způsobem je bez funkčního indexu potenciálně problematické dělat jakékoli operace (kromě operací, které se provádějí za klíčovým slovem SELECT, protože ty se provedou až na získaných datech).

Stránkování

- Stránkování pomocí LIMIT a OFFSET je problematické …
  - http://use-the-index-luke.com/blog/2013-07/pagination-done-the-postgresql-way

Znovu vytvoření indexu

- Kdy můžete chtít zrušit a znovu vytvořit stávající index?
  - Když chcete změnit charakteristiku indexu.
  - Když jste provedli nad tabulkou hodně DML příkazů. Kvůli tomu se může velikost indexu zvýšit a když máte indexy na většině sloupců tabulky, pak dokonce může být index větší než samotná tabulka!
  - Pro znovu vytvoření indexu slouží příkaz:

    ```
    ALTER INDEX <název indexu> REBUILD;
    ```

    - Je to nejrychlejší a nejefektivnější způsob.
  - Když chcete změnit charakteristiku indexu, pak ho musíte zrušit a znovu postavit:

    ```
    DROP INDEX <název indexu>;
    CREATE INDEX ...
    ```

Integritní omezení & DML – rychlost

- Jak hodně zpomalují integritní omezení DML operace?
  - Hodně :-) Tak hodně, že může být efektivnější integritní omezení vypnout / smazat, provést DML operace a poté je znovu zapnout / vytvořit.

Batch

- Při provádění hromady operací zjistíte, že je jejich provádění nějaké pomalé :-) … jak to zrychlit?
  - Například INSERT multirow VALUES:

```
INSERT INTO films (code, title, did, date_prod, kind) VALUES
    ('B6717', 'Tampopo', 110, '1985-02-10', 'Comedy'),
    ('HG120', 'The Dinner Game', 140, DEFAULT, 'Comedy');
```

  - U PL/SQL bulk-collect-forall:
    - Viz. soubor test-bulk-collect-forall.txt
  - Nebo JDBC bach:
    - https://www.tutorialspoint.com/jdbc/jdbc-batch-processing.htm

Všechny cesty vedou do Říma I.

```
-- cost = 5
select jmeno, prijmeni
from zamestnanec
left join pilot
using (zamestnanec_id)
where pilot_id is null;
```

```
-- cost = 6
select jmeno, prijmeni
from zamestnanec
minus
select jmeno, prijmeni
from zamestnanec
join pilot
using (zamestnanec_id);
```

Na těchto dvou stránkách jsou různé způsoby řešení příkladu:
Vypište všechny zaměstnance, kteří nejsou piloti.

Všechny cesty vedou do Říma II.

```
-- cost = 2
select jmeno, prijmeni
from zamestnanec
where zamestnanec_id
  not in (
    select zamestnanec_id
    from zamestnanec
    join pilot
    using (zamestnanec_id));
```

```
-- cost = 2
select jmeno, prijmeni
from zamestnanec z
where not exists (
    select 1 from pilot p
     where p.zamestnanec_id =
           z.zamestnanec_id
    );
```